(12) UK Patent Application (19) GB (11) 2 332 075 (13) A

(43) Date of A Publication 09.06.1999

(21) Application No 9725808.1

(22) Date of Filing 06.12.1997

(51) INT CL[6]
G06F 9/30

(52) UK CL (Edition Q)
G4A APM APX

(56) Documents Cited
None

(58) Field of Search
UK CL (Edition P) G4A APM APP APX
INT CL[6] G06F 9/22 9/30

(71) Applicant(s)
Mitel Corporation
(Incorporated in Canada - Ontario)
Po Box 13089, 350 Legget Drive, Kanata,
Ontario K2K 1X3, Canada

(72) Inventor(s)
Alexander Tulai

(74) Agent and/or Address for Service
Dummett Copp
25 The Square, Martlesham Heath, IPSWICH, Suffolk,
IP5 3SL, United Kingdom

(54) Abstract Title
Optimized instruction storage and distribution for parallel processor architecture

(57) A method of improving the utilization of program memory in a multi parallel processor architecture which utilizes an instruction register file (IRE). The IRE is partitioned into two pages and grouping bits are added to the program instructions to designate the fetch cycle to which the instruction belongs. Routing bits are also used to route the instructions properly to the designated processor. The relative position of the routing instruction within the set of instructions is also used to provide routing information.

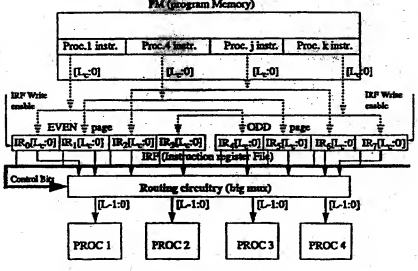

FIGURE 3

GB 2 332 075 A

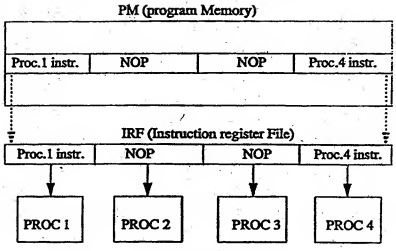

FIGURE 1

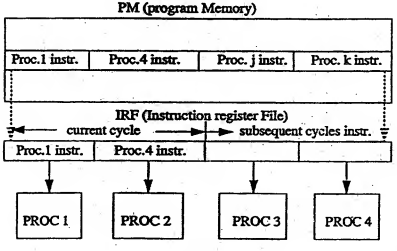

FIGURE 2

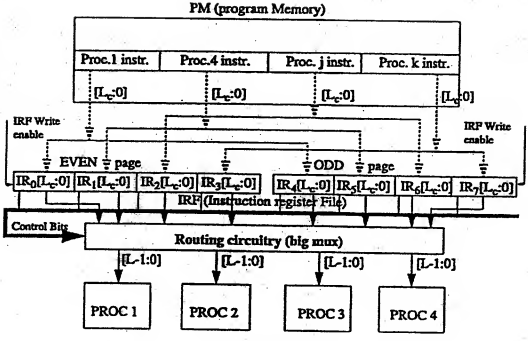
PM (program Memory)
Proc.1 instr.
Proc.4 instr.
Proc. j instr.
Proc. k instr.
[Lc:0]
IRF Write enable
EVEN page
ODD page
IR0[Lc:0]
IR1[Lc:0]
IR2[Lc:0]
IR3[Lc:0]
IR4[Lc:0]
IR5[Lc:0]
IR6[Lc:0]
IR7[Lc:0]
IRF (Instruction register File)
Control Bits
Routing circuitry (big mux)
[L-1:0]
PROC 1
PROC 2
PROC 3
PROC 4

FIGURE 3

OPTIMIZED INSTRUCTION STORAGE AND DISTRIBUTION FOR PARALLEL PROCESSORS ARCHITECTURES

Field of the Invention

This invention relates to multiple processors in a parallel configuration and more particularly to a method of improving the utilization of program memory in the process of fetching and distributing instructions when an instruction register file is used.

Background of the Invention

In a single processor architecture, the execution of a program is conventionally divided into three major phases. These phases are: instruction fetching which involves reading one instruction from the program memory into the instruction register (IR); instruction decoding which involves decoding the instructions from IR and preparing the control signals for its execution; and executing the instruction.

In a parallel processor architecture, multiple instructions have to be read from the program memory into multiple instruction registers that could be organized into an instruction register file (IRF). If there are n

processors in a multi-parallel processor architecture, there should be at least n instructions which are read from the program into the IRF if unnecessary delays are to be avoided. In multiple processor architectures, however, it is not guaranteed that all the processors will have an instruction to execute every cycle. In this case, a no operation (NOP) instruction will have to be routed to the processor for decoding and execution. Obviously, storing NOP instructions into the program memory is wasteful of program memory and ways of eliminating NOPs have been investigated.

Simply eliminating the NOP instructions creates routing problems in a multiple parallel processor architecture, as it is impossible to successfully route the instructions without additional information.

To overcome this problem it is known to add control bits to the instructions stored in the program memory for use in grouping instructions and routing information to the intended processor. The requirement to introduce grouping and routing bits to the instructions adds complexity to the system architecture and increases power requirements.

Summary of the Invention

The present invention seeks to provide better utilization of the program memory in a multi-parallel

processor implementation by allowing instructions to stretch between to consecutive instruction packs.

The present invention provides a simplified instruction distribution circuit in that the bit routing coding makes use of the instruction position within the group (for groups of two or more instructions).

In the present invention unused distribution control bits are created in certain cases and these bits may be used for additional functionality.

Therefore, in accordance with a first aspect of the present invention there is provided in a multi-parallel processor architecture having a program manager for storing processor instructions, an instruction register file for decoding instructions fetched from the program memory for execution by selected ones of the parallel processors a method of improving utilization of the program memory. The method comprises a) partitioning the instruction register file; b) providing a grouping bit to the instructions to identify the fetch cycle to which the instruction belongs and c) providing routing bits to the instructions to designate which of the processors the instruction is for, wherein the relative position of the routing bit within the instruction provides routing information.

In accordance with a second aspect of the invention there is provided a system for distributing instructions

from a program memory to multi-parallel processors comprising: a partitioned instruction register file (IRF) for receiving and decoding instructions from the program memory; multiple buses for carrying instructions from the memory to the IRF; routing circuitry for directing instructions to designated processors; and a bit route coding sequence to distribute instructions, the route coding utilizing the instruction position within the sequence to provide routing information.

10

Brief Description of the Drawings

The invention will now be described in greater detail with reference to the attached drawings wherein:
Figure 1 illustrates a multi-parallel processor architecture according to the prior art;
Figure 2 illustrates the problem caused by eliminating non-operational instructions from the set of instructions; and
Figure 3 illustrates a four-processor architecture which implements the present invention.

20

Detailed Description of the Invention

As previously discussed the execution of a program is divided in most of the processors in use today into three major phases. These are:

1) instruction fetching

2) instruction decoding; and

3) instruction execution.

In a parallel processor architecture multiple instructions have to be read from the Program Memory into multiple IRs that could be organized in a Instruction Register File (IRF).

Assuming that a certain architecture uses "n" processors, at least "n" instructions should be read from the Program Memory into the IRF if unnecessary delays are to be avoided. However in multi processor architectures it is not guaranteed that all the processors will have an instruction to execute every cycle, in which case a NOP (NO Operation) instruction will have to be routed to the processor for decoding and execution. Storing NOP instructions into the Program Memory is rather wasteful and ways of eliminating them have been sought. Figure 1 illustrates a multi parallel processor architecture in which the program memory has stored NOP instructions respecting processors 2 and 3. These NOP instructions are fetched to the Instruction Register File for delivery to the respective processors. Obviously this results in memory usage involving no exchange of meaningful data.

If the undesired NOP instructions are eliminated, the routing of the instructions is impossible without additional

information. Figure 2 illustrates the assignment problem for the case of n=4. In this example the NOP instructions relating to processors 2 and 3 have been eliminated and the program memory which would have been used for NOP instructions used for other processor instructions. As indicated in Figure 2 this results in instructions intended for processor 4 being wrongly directed to processor 2.

To solve this problem, control bits are added to the instructions stored in the Program Memory. The number of instruction goes from constant (4 in Figure 1) to variable (anywhere from 1 to 4 in the case of a 4 processor architecture or 1 to n in the general case of n processors). These control bits carry two kind of information:

1) grouping information (grouping together all the instructions that have to be executed in the same cycle but on different processors); and

2) routing information (maps an instruction to a certain processing unit).

Because of these definitions they shall be referred to as grouping control bits and routing control bits.

In such systems the issues that have to be addressed are:

1) How are the instructions stored in the Program Memory and how many of them are written into the IRF in one fetch cycle?

2) What is the optimal size of the IRF (how many

instructions can it accommodate)?

3) What configuration of control bits allows for an optimal distribution of the instruction from the IRF to the processing units?

4) How are the flow control changes (jumps, call to subroutines etc.) handled? and

5) How does the size of IRF influence the number of reads from the PM and the impact on the power consumption of the device?

The present invention demonstrates that the Program Memory waste could be further reduced and the instruction distributing circuitry could be simplified by:

1. allowing a set of instructions (that is to be executed in the same cycle) to spread over two consecutive Program Memory fetch lines;

2. dimensioning IRF to 2*n where n is a power of 2 (but not necessarily);

3. coding the distribution control bits as follows:

3.1) use $r=[\log_2(2n-1)]$ bits per instruction for routing control;

3.2) in a set of p instructions belonging to the same cycle, with $p \geq m$ (where m is the minimum integer such that $m*r \geq n$), assign each distribution control bit of the first m instructions to one of the n processors and set them to 0 or

1 to indicate which processor receives which instruction in the set of p (the matching is done positionally from left to right)

4. If the grouping control bits indicate that more than n instructions belong to the same group, do not advance the decoding pointer in IRF

5. Upon a flow control change, set all the grouping bits of the IRF, that will not be written to during the first fetch cycle, to such a value that an instruction spreading over two consecutive Program Memory locations (at the addressed jumped to) could not be falsely grouped with instruction left over in the IRF before the flow control occurred.

As mentioned above in a system with n processors, at least n instructions should be read at a time from the Program Memory (PM) if delays are to be avoided. Consequently a minimum IRF capacity of n instructions guarantees that no delays are introduced during the fetching phase.

If IRF can store more than n instructions, that would allow the elimination of the fetching phase upon jumps to locations that are already in the IRF. From this point of view a larger IRF would behave like a cache memory. However, the size of the routing circuit needed to send an instruction from IRF to the proper processor becomes huge

when any IRF register could be routed to any processor, a situation that occurs if one wants to eliminate the memory wasteful NOPs by accepting a variable instruction register. In these conditions the size of the routing circuitry is kept to a minimum and no additional delays are introduced during the fetching phase if the capacity of the IRF is set to exactly n instructions. However, a third factor in deciding the size of the IRF is the waste of program memory location that occurs when the size of the IRF is exactly n and the instructions to be decoded and executed every cycle is variable (anywhere from 1 to n).

When variable sized instructions are packed in groups of n and stored in the program memory, it could happen that the room left in the current pack is not enough to fit the next instruction in which case the rest of the pack will be filled with NOPs and a new pack started. The worst scenario possible is that (n-1) locations are available in the current pack while the next cycle instruction length is exactly n. In such a case the waste could be as high as (n- 1) instructions. The best case is obviously when instructions could be fitted exactly in an n instruction pack.

To address this waste, we could allow an instruction to stretch between two packs of length n and thus eliminate any waste of PM locations. However, this feature requires the

extension of the IRF capacity from n to 2*n.

A pointer within IRF will indicate where the next instruction to be decoded starts. When this points to an instruction that starts in one pack and finishes in the next pack, the instruction cannot be decoded unless the rest of it is fetched from the PM and available in the IRF. That's why doubling the size of the IRF from n to 2*n solves the problem as we could alternatively fetch in one half of IRF or the other and when an instruction that stretches between two consecutive packs is to be decoded both the beginning and the end of the instructions are found in IRF. Wrap arounding is used in such a case to maintain the continuity of an instruction.

Having two pages of n instructions significantly increases the size of the IRF circuitry and that of the routing circuitry, however, considerable program memory savings are made possible (some examples on a 4 processor architecture have shown savings of up to 20% for certain programs). In addition to this, 2*n locations are enough for the code for some tight loops, the kind we encounter in filtering, to be fully stored in IRF and that would avoid program memory fetches during filtering and consequently would reduce the overall power consumption of the chip. Considering these advantages an IRF with a capacity of 2*n is optimal. Increasing the capacity of the IRF beyond 2*n

instructions could reduce the power consumption in certain cases and that for a very high cost in increased IRF and routing circuitry, and it is not justifiable in general.

The control bits used for grouping are used to indicate which instructions from IFR should be routed to the n processors for decoding during the current cycle. The minimum number of bits used for this operation for each instruction is 1. The routing circuitry will analyse the grouping bit for n consecutive IRF instructions and the decisions taken are summarized in Table 1.

Table 1: Grouping control bits decoder[a]

	Instr. 1	Instr. 2	.....	Instr. n	Decision
GROUPING BIT	p	x	x.....x	x	1 instruction cycle
	$\bar{p}$	p	x.....x	x	2 instruction cycle
	$\bar{p}$	$\bar{p}$	p x.....x	x	3 instruction cycle
	.	.	.	.	And so on
	.	.	.	.	
	$\bar{p}$	$\bar{p}$	$\bar{p}$.....$\bar{p}$p	x	n-1 instruction cycle
	$\bar{p}$	$\bar{p}$	$\bar{p}$.....$\bar{p}$	p	n instruction cycle
	$\bar{p}$	$\bar{p}$	$\bar{p}$.....$\bar{p}$	$\bar{p}$	0 instruction cycle, NOps will be pushed to all processors

a. x - don't care, p - 0/1 , $\bar{p}$ - 1/0

r=[log2(2n-1)] bits are required to identify to what processor an individual instruction should be routed, where [] is the integer part function defined as:

$$[X] = \begin{cases} x, x \in N \\ n, n<x<n+1, n \in N \end{cases}$$

However, if each instruction carries its own routing bits, a redundancy appears when groups of p instructions with p > m, where m is such that m * r ≥ n, do not exploit the position of the instruction in the group.

Consider the example where n=4.

The number of routing bits r required for routing one instruction is:

r=log2(2*4-1)]=[2.8]=2 which indeed corresponds to the 4 possible combinations one can make with two bits.

For m=2 we have: m*r=4=n so for any group of p > 2 instructions we have some redundancy within the routing bits if the position of the instruction in the group is not exploited.

For p=3,r=2,n=4 and the following three instructions:

$$\bar{g}r_0^1 r_1^1 i_1^1 i_2^1 \ldots i_L^1 \quad \bar{g}r_0^2 r_1^2 i_1^2 i_2^2 \ldots i_L^2 \quad g r_0^3 r_1^3 i_1^3 i_2^3 \ldots i_L^3$$

where:

g's are the grouping bits

r's are the routing bits

i's are the instruction bits

L is the instruction length

and let's assume as well that the 3 instructions should go to processors 1, 3 and 4 with g=1 and the natural assignment of the 2 bit combinations we have the following control bits for the example given:

$$000i_1^1 i_2^1 \ldots i_L^1 \qquad 010i_1^2 i_2^2 \ldots i_L^2 \qquad 111i_1^3 i_2^3 \ldots i_L^3$$

However, the following group of identical instructions would be distributed just as well to the processors 1, 3 and 4 for the simple reason that each instruction carries its own routing bits and the order of the instructions in the group doesn't count

$$011i_1^3 i_2^3 \ldots i_L^3 \qquad 000i_1^1 i_2^1 \ldots i_L^1 \qquad 110i_1^2 i_2^2 \ldots i_L^2$$

This introduces a redundancy that translates into a somewhat faster circuit but significantly larger than in the case when the position of the instructions in the group would be exploited.

If we assume that the three instructions to be routed to processors 1, 3 and 4 are placed exactly in this order when packed and placed in the program memory, we would need exactly n=4 bits to show how the mapping is done.

By concatenating the routing bits of the first two instructions ($r_0^1 r_1^1$ and $r_0^2 r_1^2$) we get exactly the 4 bits needed to show how the assignment is done and the following table will cover all possible cases for groups of 2, 3 and 4 instructions.

Table 2: Proposed routing bits assignment

p	$r_0^1 r_1^1 r_0^2 r_1^2$	Routing decision
2	0011	NOP->proc.1,NOP->proc.2, $i_1^1 i_2^1 \ldots i_L^1$ ->proc.3 $i_1^2 i_2^2 \ldots i_L^2$ ->proc.4
2	0101	NOP->proc.1, $i_1^1 i_2^1 \ldots i_L^1$ ->proc.2,NOP->proc.3 $i_1^2 i_2^2 \ldots i_L^2$ ->proc.4
2	1001	$i_1^1 i_2^1 \ldots i_L^1$ ->proc.1,NOP->proc.2,NOP->proc.3 $i_1^2 i_2^2 \ldots i_L^2$ ->proc.4
2	0110	NOP->proc.1, $i_1^1 i_2^1 \ldots i_L^1$ ->proc.2, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.3 NOP->proc.4
2	1010	$i_1^1 i_2^1 \ldots i_L^1$ ->proc.1,NOP->proc.2, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.3 NOP->proc.4
2	1100	$i_1^1 i_2^1 \ldots i_L^1$ ->proc.1, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.2,NOP->proc.3 NOP->proc.4
3	0111	NOP->proc.1, $i_1^1 i_2^1 \ldots i_L^1$ ->proc.2, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.3 $i_1^3 i_2^3 \ldots i_L^3$ ->proc.4
3	1011	$i_1^1 i_2^1 \ldots i_L^1$ ->proc.1,NOP->proc.2, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.3 $i_1^3 i_2^3 \ldots i_L^3$ ->proc.4
3	1101	$i_1^1 i_2^1 \ldots i_L^1$ ->proc.1, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.2,NOP->proc.3 $i_1^3 i_2^3 \ldots i_L^3$ ->proc.4
3	1110	$i_1^1 i_2^1 \ldots i_L^1$ ->proc.1, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.2, $i_1^3 i_2^3 \ldots i_L^3$ ->proc.3, NOP->proc.4
4	1111	$i_1^1 i_2^1 \ldots i_L^1$ ->proc.1, $i_1^2 i_2^2 \ldots i_L^2$ ->proc.2, $i_1^3 i_2^3 \ldots i_L^3$ ->proc.3, $i_1^4 i_2^4 \ldots i_L^4$ ->proc.4

Depending how the circuit is implemented the last coding could prove redundant just as well because for the case p=4 it is clear which instruction goes to which

processor (because we have an equal number of instructions and processors).

Going back to the previous example, we see now that bits $r_0r_1^{3\,3}$ are not used any more. These bits are redundant. In the case of 4 instructions in the group not only are two more bits becoming redundant ($r_0r_1^{4\,4}$) but depending on the implementation even the first four bits ($r_0r_1^{1\,1}$ and $r_0r_1^{2\,2}$) could be redundant.

Because the routing of the instructions from the IRF to the processors is done after the instructions have been fetched from the program memory into the IRF, at least one cycle of delay will be introduced during an instruction flow change to a PM location that is not already loaded into IRF.

During this time NOPs will be pushed to all processors. However if the instruction at the address jumped to, is one that stretches over two consecutive packs of n instructions, an additional cycle is needed to load the second pack into the second IRF page, before the instruction could actually be routed to the appropriate processor.

However, because of the previous instructions left over in that IRF page, an early and wrong routing might take place, if during a jump, the second page of the IRF (first in this case being the one to which the first PM location is fetched into) will have all its grouping bits set to g such that the decoder will be forced to default to the last case

in Table 1, with NOPs being pushed to all processors and the pointer within IRF preserving the old value.

This is a very elegant way of handling the jumps because it does not require any additional circuitry. Moreover the circuit will work just as well during RESET when all the grouping bits will be held to $\overline{g}$ and NOPs will be pushed automatically to all units while the pointer will be locked at 0.

Figure 3 shows an architecture with 4 processors, 3 control bits (one for grouping and two for routing), an IRF with two pages of 4 instructions each. Not shown in Figure 3 is the circuitry that enables writing to IRF, based on the value of the current IRF pointer, the grouping and control bits and the instructions to be executed. Figure 3 does show a 4 processor implementation wherein the IRF has two pages of four instructions each. From the program memory, four buses carry four instructions to IRF during a fetch cycle. Each bus is L_e+1 bits wide with three control bits and L instruction bits, so that L_e= L+2.

Although one implementation of the invention has been described and illustrated it will be apparent to one skilled in the art that several alterations can be made without departing from the basic concept. It is to be understood that such alterations will fall within the scope of the invention as defined by the appended claims.

Claims:

1. In a multi, parallel processor architecture having a program memory for storing processor instructions and an instruction register file for decoding instructions fetched from said program memory for execution by selected ones of said parallel processors, a method of improving utilization of said program memory comprising: a) partitioning said instruction register file; b) providing a grouping bit to said instructions to identify the fetch cycle to which said instruction belongs; and c) providing routing bits to said instructions to designate which of said processors said instruction is to be routed, wherein the position of said routing bit within said instructions provides routing information.

2. A method as defined in claim 1, wherein said instruction register file is partitioned into two sections.

3. A method as defined in claim 2 wherein the number of parallel processors is n and the capacity of the instruction register file is 2n.

4. A method as defined in claim 3 wherein the number of routing bits (r) is in accordance with the expression: $r=[\log_2(2n-1)]$.

5. A method as defined in claim 1 wherein said grouping bit is used to indicate which instructions from the instruction register file is to be decoded in the current cycle.

6. A system for distributing instructions from a program memory to multi-parallel processors comprising:
a partitioned instruction register file (IRF) for receiving and decoding instructions from the program memory;
multiple buses for carrying instructions from the memory to the IRF;
routing circuitry for directing instructions to designated processors; and
a bit route coding sequence to distribute instructions, the route coding utilizing the instruction position within the sequence to provide routing information.

7. A system as defined in claim 6 wherein said instruction register file is partitioned into two pages.

8. A system substantially as herein described, with reference to figure 3 of the accompanying drawings.

The Patent Office

Application No: GB 9725808.1
Claims searched: 1,6

Examiner: Leslie Middleton
Date of search: 17 June 1998

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.P): G4A (APM, APP, APX)

Int Cl (Ed.6): G06F 9/30

Other: Online: COMPUTER DATABASE, INSPEC, WPI.

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
	NONE	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.